下载APP

05 | 内存快照：宕机后，Redis如何实现快速恢复？

2020-08-14 蒋德钧

Redis核心技术与实战

进入课程 >

讲述：蒋德钧
时长 14:58　大小 13.72M

你好，我是蒋德钧。

上节课，我们学习了 Redis 避免数据丢失的 AOF 方法。这个方法的好处，是每次执行只需要记录操作命令，需要持久化的数据量不大。一般而言，只要你采用的不是 always 的持久化策略，就不会对性能造成太大影响。

但是，也正因为记录的是操作命令，而不是实际的数据，所以，用 AOF 方法进行故障恢复的时候，需要逐一把操作日志都执行一遍。如果操作日志非常多，Redis 就会恢复得很缓慢，影响到正常使用。这当然不是理想的结果。那么，还有没有既可以保证可靠性，还在宕机时实现快速恢复的其他方法呢？

当然有了，这就是我们今天要一起学习的另一种持久化方法：**内存快照**。所谓内存快照，就是指内存中的数据在某一个时刻的状态记录。这就类似于照片，当你给朋友拍照时，一张照片就能把朋友一瞬间的形象完全记下来。

对 Redis 来说，它实现类似照片记录效果的方式，就是把某一时刻的状态以文件的形式写到磁盘上，也就是快照。这样一来，即使宕机，快照文件也不会丢失，数据的可靠性也就得到了保证。这个快照文件就称为 RDB 文件，其中，RDB 就是 Redis DataBase 的缩写。

和 AOF 相比，RDB 记录的是某一时刻的数据，并不是操作，所以，在做数据恢复时，我们可以直接把 RDB 文件读入内存，很快地完成恢复。听起来好像很不错，但内存快照也并不是最优选项。为什么这么说呢？

我们还要考虑两个关键问题：

对哪些数据做快照？这关系到快照的执行效率问题；

做快照时，数据还能被增删改吗？这关系到 Redis 是否被阻塞，能否同时正常处理请求。

这么说可能你还不太好理解，我还是拿拍照片来举例子。我们在拍照时，通常要关注两个问题：

如何取景？也就是说，我们打算把哪些人、哪些物拍到照片中；

在按快门前，要记着提醒朋友不要乱动，否则拍出来的照片就模糊了。

你看，这两个问题是不是非常重要呢？那么，接下来，我们就来具体地聊一聊。先说“取景”问题，也就是我们对哪些数据做快照。

给哪些内存数据做快照？

Redis 的数据都在内存中，为了提供所有数据的可靠性保证，它执行的是**全量快照**，也就是说，把内存中的所有数据都记录到磁盘中，这就类似于给 100 个人拍合影，把每一个人都拍进照片里。这样做的好处是，一次性记录了所有数据，一个都不少。

当你给一个人拍照时，只用协调一个人就够了，但是，拍 100 人的大合影，却需要协调 100 个人的位置、状态，等等，这当然会更费时费力。同样，给内存的全量数据做快照，把它们全部写入磁盘也会花费很多时间。而且，全量数据越多，RDB 文件就越大，往磁盘上写数据的时间开销就越大。

对于 Redis 而言，它的单线程模型就决定了，我们要尽量避免所有会阻塞主线程的操作，所以，针对任何操作，我们都会提一个灵魂之问：“它会阻塞主线程吗?”RDB 文件的生成是否会阻塞主线程，这就关系到是否会降低 Redis 的性能。

Redis 提供了两个命令来生成 RDB 文件，分别是 save 和 bgsave。

save：在主线程中执行，会导致阻塞；

bgsave：创建一个子进程，专门用于写入 RDB 文件，避免了主线程的阻塞，这也是 Redis RDB 文件生成的默认配置。

好了，这个时候，我们就可以通过 bgsave 命令来执行全量快照，这既提供了数据的可靠性保证，也避免了对 Redis 的性能影响。

接下来，我们要关注的问题就是，在对内存数据做快照时，这些数据还能“动”吗? 也就是说，这些数据还能被修改吗？ 这个问题非常重要，这是因为，如果数据能被修改，那就意味着 Redis 还能正常处理写操作。否则，所有写操作都得等到快照完了才能执行，性能一下子就降低了。

快照时数据能修改吗?

在给别人拍照时，一旦对方动了，那么这张照片就拍糊了，我们就需要重拍，所以我们当然希望对方保持不动。对于内存快照而言，我们也不希望数据“动”。

举个例子。我们在时刻 t 给内存做快照，假设内存数据量是 4GB，磁盘的写入带宽是 0.2GB/s，简单来说，至少需要 20s（4/0.2 = 20）才能做完。如果在时刻 t+5s 时，一个还没有被写入磁盘的内存数据 A，被修改成了 A’ ，那么就会破坏快照的完整性，因为 A’ 不是时刻 t 时的状态。因此，和拍照类似，我们在做快照时也不希望数据“动” ，也就是不能被修改。

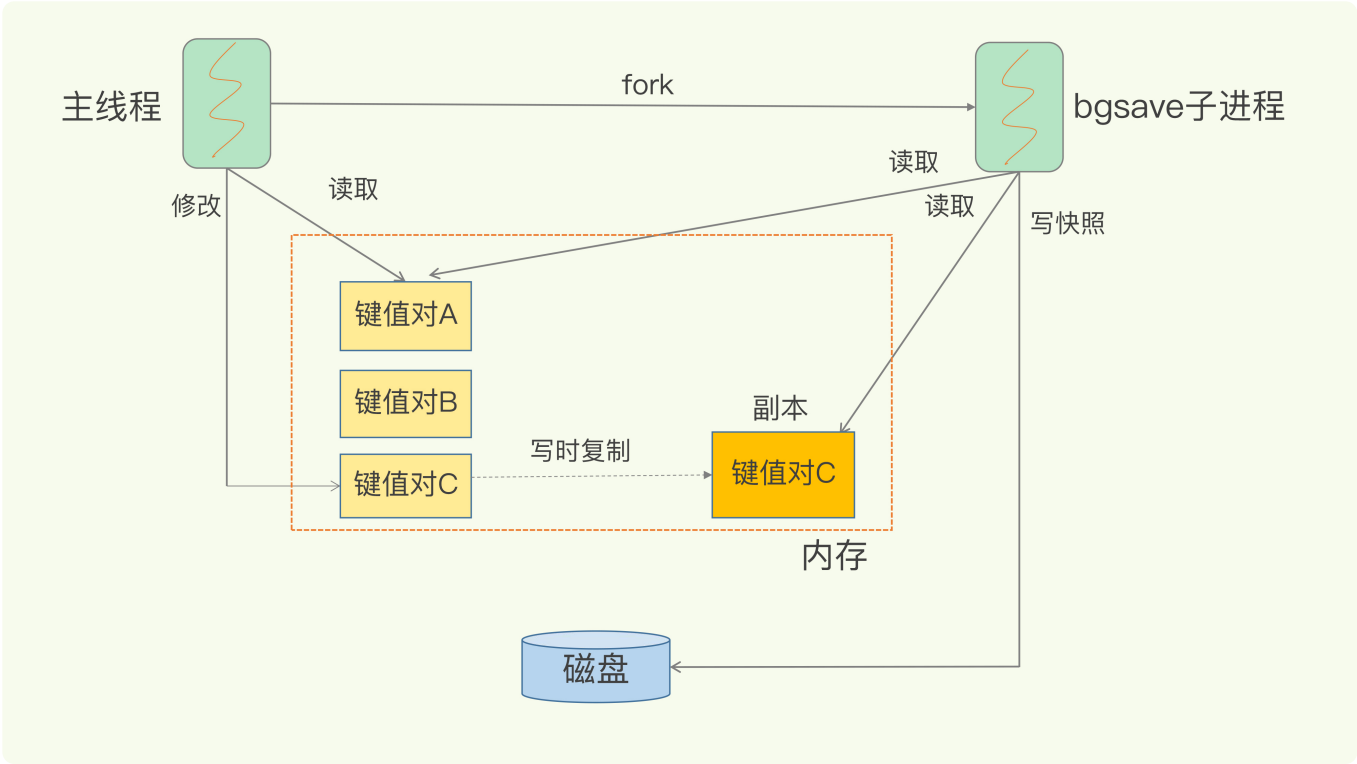
但是，如果快照执行期间数据不能被修改，是会有潜在问题的。对于刚刚的例子来说，在做快照的 20s 时间里，如果这 4GB 的数据都不能被修改，Redis 就不能处理对这些数据的写操作，那无疑就会给业务服务造成巨大的影响。

你可能会想到，可以用 bgsave 避免阻塞啊。这里我就要说到一个常见的误区了，**避免阻塞和正常处理写操作并不是一回事**。此时，主线程的确没有阻塞，可以正常接收请求，但是，为了保证快照完整性，它只能处理读操作，因为不能修改正在执行快照的数据。

为了快照而暂停写操作，肯定是不能接受的。所以这个时候，Redis 就会借助操作系统提供的写时复制技术（Copy-On-Write, COW），在执行快照的同时，正常处理写操作。

简单来说，bgsave 子进程是由主线程 fork 生成的，可以共享主线程的所有内存数据。bgsave 子进程运行后，开始读取主线程的内存数据，并把它们写入 RDB 文件。

此时，如果主线程对这些数据也都是读操作（例如图中的键值对 A），那么，主线程和 bgsave 子进程相互不影响。但是，如果主线程要修改一块数据（例如图中的键值对 C），那么，这块数据就会被复制一份，生成该数据的副本。然后，bgsave 子进程会把这个副本数据写入 RDB 文件，而在这个过程中，主线程仍然可以直接修改原来的数据。



写时复制机制保证快照期间数据可修改

这既保证了快照的完整性，也允许主线程同时对数据进行修改，避免了对正常业务的影响。

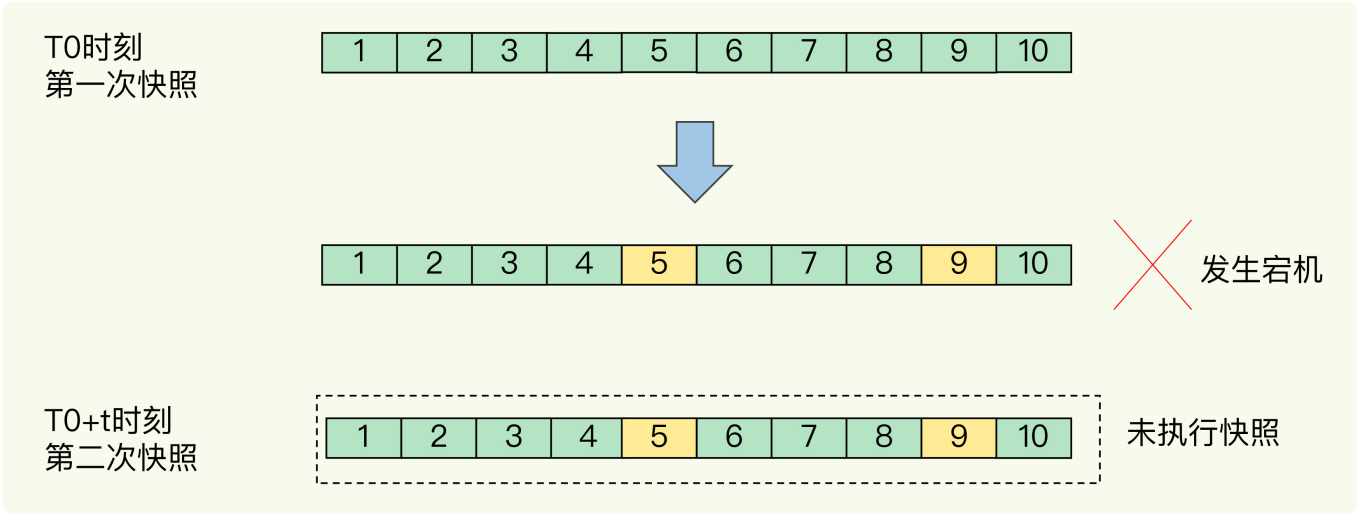
到这里，我们就解决了对“哪些数据做快照”以及“做快照时数据能否修改”这两大问题：Redis 会使用 bgsave 对当前内存中的所有数据做快照，这个操作是子进程在后台完成的，这就允许主线程同时可以修改数据。

现在，我们再来看另一个问题：多久做一次快照？我们在拍照的时候，还有项技术叫“连拍”，可以记录人或物连续多个瞬间的状态。那么，快照也适合“连拍”吗？

可以每秒做一次快照吗？

对于快照来说，所谓“连拍”就是指连续地做快照。这样一来，快照的间隔时间变得很短，即使某一时刻发生宕机了，因为上一时刻快照刚执行，丢失的数据也不会太多。但是，这其中的快照间隔时间就很关键了。

如下图所示，我们先在 T0 时刻做了一次快照，然后又在 T0+t 时刻做了一次快照，在这期间，数据块 5 和 9 被修改了。如果在 t 这段时间内，机器宕机了，那么，只能按照 T0 时刻的快照进行恢复。此时，数据块 5 和 9 的修改值因为没有快照记录，就无法恢复了。



快照机制下的数据丢失

所以，要想尽可能恢复数据，t 值就要尽可能小，t 越小，就越像“连拍”。那么，t 值可以小到什么程度呢，比如说是不是可以每秒做一次快照？毕竟，每次快照都是由 bgsave 子进程在后台执行，也不会阻塞主线程。

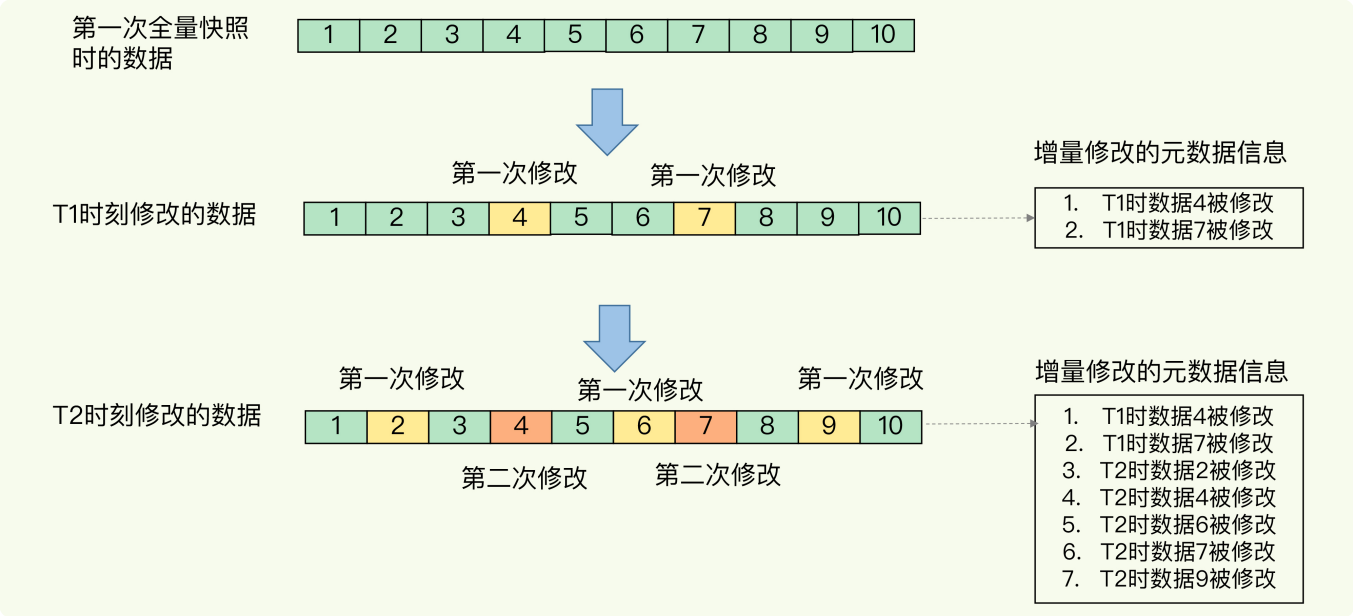
这种想法其实是错误的。虽然 bgsave 执行时不阻塞主线程，但是，**如果频繁地执行全量快照，也会带来两方面的开销。**

一方面，频繁将全量数据写入磁盘，会给磁盘带来很大压力，多个快照竞争有限的磁盘带宽，前一个快照还没有做完，后一个又开始做了，容易造成恶性循环。

另一方面，bgsave 子进程需要通过 fork 操作从主线程创建出来。虽然，子进程在创建后不会再阻塞主线程，但是，fork 这个创建过程本身会阻塞主线程，而且主线程的内存越大，阻塞时间越长。如果频繁 fork 出 bgsave 子进程，这就会频繁阻塞主线程了。那么，有什么其他好方法吗？

此时，我们可以做增量快照，所谓增量快照，就是指，做了一次全量快照后，后续的快照只对修改的数据进行快照记录，这样可以避免每次全量快照的开销。

在第一次做完全量快照后，T1 和 T2 时刻如果再做快照，我们只需要将被修改的数据写入快照文件就行。但是，这么做的前提是，**我们需要记住哪些数据被修改了**。你可不要小瞧这个“记住”功能，它需要我们使用额外的元数据信息去记录哪些数据被修改了，这会带来额外的空间开销问题。如下图所示：



增量快照示意图

如果我们对每一个键值对的修改，都做个记录，那么，如果有 1 万个被修改的键值对，我们就需要有 1 万条额外的记录。而且，有的时候，键值对非常小，比如只有 32 字节，而

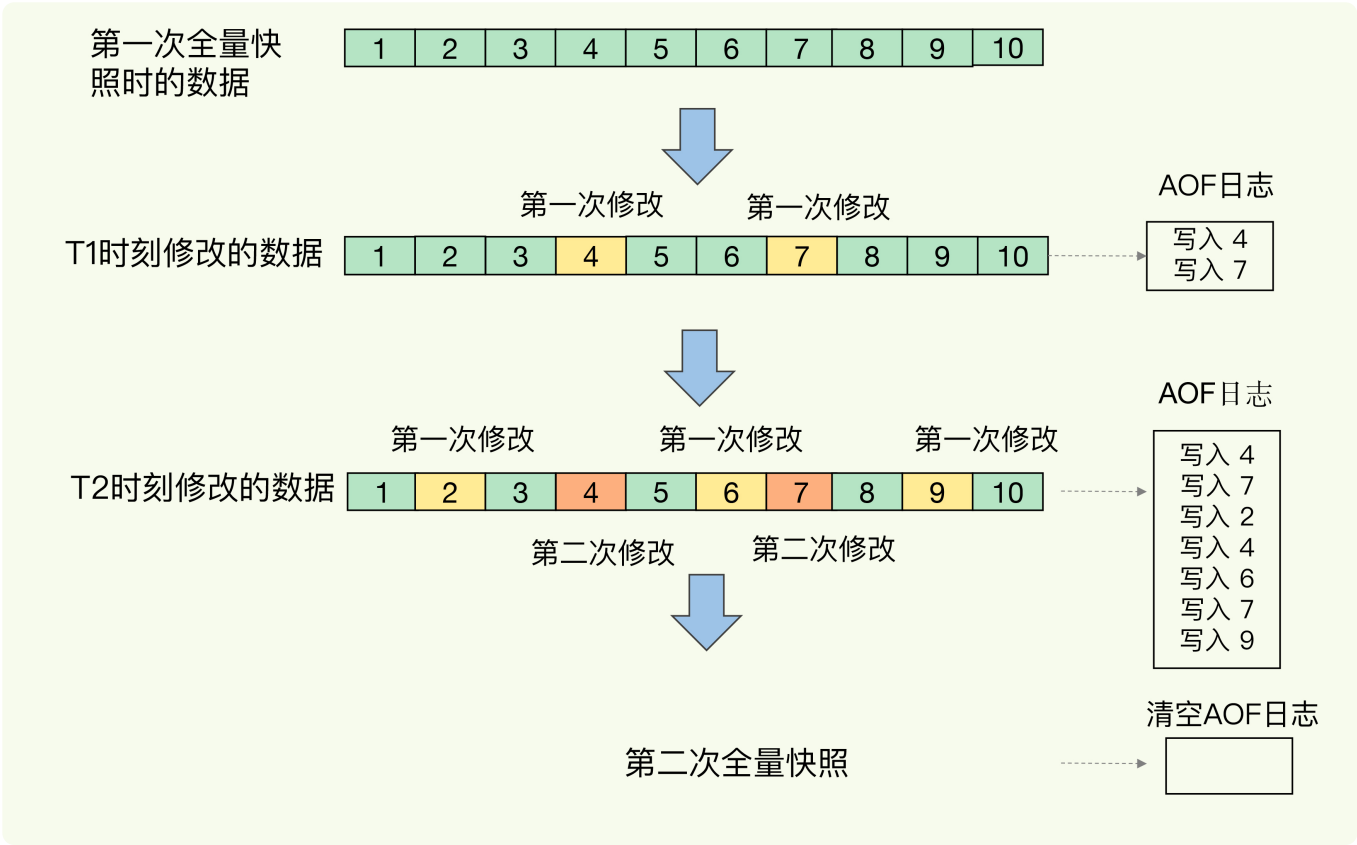
记录它被修改的元数据信息，可能就需要 8 字节，这样的画，为了“记住”修改，引入的额外空间开销比较大。这对于内存资源宝贵的 Redis 来说，有些得不偿失。

到这里，你可以发现，虽然跟 AOF 相比，快照的恢复速度快，但是，快照的频率不好把握，如果频率太低，两次快照间一旦宕机，就可能有比较多的数据丢失。如果频率太高，又会产生额外开销，那么，还有什么方法既能利用 RDB 的快速恢复，又能以较小的开销做到尽量少丢数据呢？

Redis 4.0 中提出了一个**混合使用 AOF 日志和内存快照**的方法。简单来说，内存快照以一定的频率执行，在两次快照之间，使用 AOF 日志记录这期间的所有命令操作。

这样一来，快照不用很频繁地执行，这就避免了频繁 fork 对主线程的影响。而且，AOF 日志也只用记录两次快照间的操作，也就是说，不需要记录所有操作了，因此，就不会出现文件过大的情况了，也可以避免重写开销。

如下图所示，T1 和 T2 时刻的修改，用 AOF 日志记录，等到第二次做全量快照时，就可以清空 AOF 日志，因为此时的修改都已经记录到快照中了，恢复时就不再用日志了。



内存快照和AOF混合使用

这个方法既能享受到 RDB 文件快速恢复的好处，又能享受到 AOF 只记录操作命令的简单优势，颇有点“鱼和熊掌可以兼得”的感觉，建议你在实践中用起来。

小结

这节课，我们学习了 Redis 用于避免数据丢失的内存快照方法。这个方法的优势在于，可以快速恢复数据库，也就是只需要把 RDB 文件直接读入内存，这就避免了 AOF 需要顺序、逐一重新执行操作命令带来的低效性能问题。

不过，内存快照也有它的局限性。它拍的是一张内存的“大合影”，不可避免地会耗时耗力。虽然，Redis 设计了 bgsave 和写时复制方式，尽可能减少了内存快照对正常读写的影响，但是，频繁快照仍然是不太能接受的。而混合使用 RDB 和 AOF，正好可以取两者之长，避两者之短，以较小的性能开销保证数据可靠性和性能。

最后，关于 AOF 和 RDB 的选择问题，我想再给你提三点建议：

数据不能丢失时，内存快照和 AOF 的混合使用是一个很好的选择；

如果允许分钟级别的数据丢失，可以只使用 RDB；

如果只用 AOF，优先使用 everysec 的配置选项，因为它在可靠性和性能之间取了一个平衡。

每课一问

我曾碰到过这么一个场景：我们使用一个 2 核 CPU、4GB 内存、500GB 磁盘的云主机运行 Redis，Redis 数据库的数据量大小差不多是 2GB，我们使用了 RDB 做持久化保证。当时 Redis 的运行负载以修改操作为主，写读比例差不多在 8:2 左右，也就是说，如果有 100 个请求，80 个请求执行的是修改操作。你觉得，在这个场景下，用 RDB 做持久化有什么风险吗？你能帮着一起分析分析吗？

到这里，关于持久化我们就讲完了，这块儿内容是熟练掌握 Redis 的基础，建议你一定好好学习下这两节课。如果你觉得有收获，希望你能帮我分享给更多的人，帮助更多人解决持久化的问题。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 04 | AOF日志：宕机了，Redis如何避免数据丢失？

下一篇 06 | 数据同步：主从库如何实现数据一致？

精选留言 (27)

写留言

Kaito 置顶

2020-08-14

2核CPU、4GB内存、500G磁盘，Redis实例占用2GB，写读比例为8:2，此时做RDB持久化，产生的风险主要在于 CPU资源 和 内存资源 这2方面：

a、内存资源风险：Redis fork子进程做RDB持久化，由于写的比例为80%，那么在持久化过程中，“写实复制”会重新分配整个实例80%的内存副本，大约需要重新分配1.6...

展开

作者回复: 除了考虑了内存风险，还考虑了CPU风险。赞！先置个顶 :D

关于绑核的操作，后面再和Kaito同学聊聊，绑核也有些值得探讨的地方的。

18 89

Darren
2020-08-14

Kaito的回答为啥老让我觉得自己那么菜呢🙈🙈🙈🙈🙈

我稍微补充下老师对于 " 混合使用 AOF 日志和内存快照 "这块的东西：
在redis4.0以前，redis AOF的重写机制是指令整合（老师上一节课已经说过），但是在redis4.0以后，redis的 AOF 重写的时候就直接把 RDB 的内容写到 AOF 文件开头，将增...
展开

4 11

注定非凡
2020-08-15

1，作者讲了什么？
作者在本章讲了redis两种持久化方式中的RDB方式
2，作者是怎么把这事给讲明白的？
为了让大家明白RDB的快照的概念，作者举了拍照片，照合影的例子
3，为了讲明白，作者讲了哪些要点，有哪些亮点？...
展开

5

扩散性百万咸面包
2020-08-14

很奇怪，对于RDB和AOF混合搭配的策略，为什么不把AOF应用于RDB生成增量快照呢？而非要再次生成全量快照？
展开

3 3

曾轼麟
2020-08-17

【内存风险】：2 核 CPU、4GB 内存、500GB 磁盘， 2GB数据，在操作系统虚拟内存的支持下，fork出一个子进程会贡献主进程的虚拟内存映射的物理空间，这个是MMU实现的不属于Redis的产物，但是当发生数据修改的时候，MMU会将子进程的物理内存复制独立出来（写时拷贝技术）。在 8:2的独写比例中实际需要的物理内存可能会达到 1.6 +1.6 =

3.2 。假设开启swap的情况下，在物理内存不能满足程序运行的时候，虚拟内存技术会...

展开

1

test

2020-08-14

1.写太多，COW需要复制的东西太多，内存占用问题；
2.CPU太少，redis后台还有很多线程在后台工作，会产生CPU竞争。

1

漫步oo0云端

2020-08-14

我想提一个傻问题，我作为初学者想问，如果redis服务挂了，备份有什么用？能恢复的前提不是服务还存活吗？难道服务挂了会自动拉起服务？自动还原吗？

展开

作者回复: 如果服务挂了，我们可以让Redis实例自动重启。此时，如果没有数据备份的话，再启动时，所有的数据都需要重新写入，这个过程会比较耗时。而如果有备份的话，Redis再启动后，可以直接读入备份数据，对于这些数据的读写操作就可以很快服务了。不知道有没有解答你的疑惑。

愿意提出来的问题都是好问题哈 :)

10 1

akka

2020-08-17

老师，文章最后你说的方案“Redis 4.0 中提出了一个混合使用 AOF 日志和内存快照的方法”，
就是redis 4.0 的AOF重写？
但是从描述上来看，各个操作的顺序和AOF重写又有点不像

展开

riryoutexi

2020-08-17

请问蒋老师，线上一般通过什么策略或依据去配置备份方式。比如 数据不能丢失时，多久 RDB 一次？

每天晒白牙
2020-08-17

Redis持久化
AOF
AOF是一种通过记录操作命令的的方式来达到持久化的目的，正是因为记录操作命令而不是数据，所以在恢复数据的时候需要 redo 这些操作命令(当然 AOF 也有重写机制从而使命令减少)，如果操作日志太多，恢复过程就会很慢，可能会影响 Redis 的正常使用...
展开

Geek_Lin
2020-08-16

文章中写时复制那里，复制的是主线程修改之前的数据还是主线程修改之后的呢？

u
2020-08-16

写时复制？那20%读操作相关的数据不就没法RDB了么？

Jackey
2020-08-16

评论中有大神，看完文章再看评论收获更多
展开

Lionheart
2020-08-15

BGSAVE生成新的RDB文件，替换老的RDB文件，这一步是怎么实现的？

yyl
2020-08-15

提问：
1. RDB文件损坏的话，是否影响Redis数据恢复？是否有人工干预方法？
2. fork进程，父子进程遵循写时复制，为什么父进程内存越大，阻塞时间越长呢？
展开

yyl
2020-08-15

解答：
系统的QPS未知，两种情况吧：
1. QPS较低，不会有什么问题
2. QPS较高，首先由于写多读少，造成更多的写时拷贝，导致更多的内存占用。如果采用增量快照，需要增加额外的内存开销；再则，写RDB文件，OS会分配一些Cache用于磁...
展开

作者回复: 非常赞！考虑到了根据不同的QPS进行分析。

我再提一个维度，你可以考虑下，就是修改的键值对的范围，也就是说写操作是针对一小部分键值对，还是针对大部分键值对的。你觉得这个维度会有影响么？

1

第四范式
2020-08-15

临时表内容太多。会浪费内存资源。这个问题。怎么解呢？老师也没说解决方案

游弋云端
2020-08-14

本质上与ZooKeeper的数据全内存、全量内存快照dump以及日志恢复机制非常相似，理论上殊途同归。

Geek_cc6b96
2020-08-14

RDB和AOF 不就是流媒体编码中的 I帧、P帧 概念嘛
展开

Spring4J
2020-08-14

由于修改操作占大部分比例，为了尽可能保证宕机时数据的完整性，快照的间隔就不能太长，而间隔太短又会带来很多的性能开销，所以对于这种特点的数据，不适合使用RDB的持久化方式

作者回复: 不错，不错！可以再想想看，在问题里的云主机上，哪怕我们先不考虑快照频率问题，单就做一次快照本身，是否可能还会有其他问题？：）